A Reconfigurable Multiprocessor IC for Rapid Prototyping of Algorithmic-Specific High-Speed DSP Data Paths

Dev C. Chen, *Member, IEEE*, and Jan M. Rabaey, *Member, IEEE*

Abstract—**A field-programmable multiprocessor integrated circuit, PADDI (for Programmable Arithmetic Devices for High-Speed Digital Signal Processing), has been designed for the rapid prototyping of high-speed data paths typical to real-time digital signal processing applications. The processor architecture addresses the key requirements of these data paths: a) fast, concurrently operating, multiple arithmetic units, b) conflict-free data routing, c) moderate hardware multiplexing (of the arithmetic units), d) minimal branch penalty between loop iterations, e) wide instruction bandwidth, and f) wide I/O bandwidth. The initial version contains eight processors connected via a dynamically controlled crossbar switch, and has a die size of 8.9 × 9.5 mm², in a 1.2-μm CMOS technology. With a maximum clock rate of 25 MHz, it can support a computation rate of 200 MIPs and can sustain a data I/O bandwidth of 400 megabytes/s with a typical power consumption of 0.45 W. An assembler and simulator have been developed to facilitate programming and testing of the chip. A software compilation path from the high-level data flow language SILAGE [15] to PADDI is currently under development, and handles partitioning, scheduling, and code generation.**

I. Introduction

In many real-time digital signal processing systems, tasks are computation intensive because high throughput is required, e.g., in real-time image processing and video applications, or because the tasks are complex, as in real-time speech recognition. Traditional microprocessor-based architectures are often inadequate to meet the computation requirements, and so clusters of dedicated data paths, hard-wired to closely match the algorithmic data flow, are used instead. Such architectures typically contain multiple and concurrently operating data-path pipelines.

The problem that we address is the rapid prototyping of computation-intensive DSP data paths. In real-time DSP applications the typical ASIC solutions take a long time to fabricate and test, and are not easily modified. A rapid prototyping capability will help alleviate these problems. In this paper we will first discuss the architectural features and computational requirements of high-speed DSP data paths. We propose a novel approach for synthesizing these data paths, first discussed in [5]. We will describe the architecture, circuit design, and implementation of a prototype chip, PADDI (for Programmable Arithmetic Devices for High-Speed Digital Signal Processing), which serves as proof of concept. We will give a brief overview of the grammar, assembler, and simulator which have been developed to assist the low-level programming of PADDI, and the CAD environment and tools being developed for automatic compilation from a high-level language. We will also compare our approach with existing ones.

Manuscript received April 26, 1992; revised July 25, 1992. This work was supported by DARPA and Sharp Microelectronics Technology, Inc.

D. C. Chen was with the Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA 94720. He is now with SUN Microsystems Inc., Mountain View, CA 94043-1100.

J. M. Rabaey is with the Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA 94720.

IEEE Log Number 9203489.

A. Computation Requirements of High-Speed DSP

The goal of this work is to define a set of high-level programmable macro components to support the rapid prototyping of real-time DSP data paths. Case studies of real-time algorithms and pipelined data-path architectures enable us to identify the following key architectural features which must be supported by these macro components:

a) a set of concurrently operating execution units (EXU's) with fast arithmetic, to satisfy the high computational (hundreds of MOPs) requirements;
b) very flexible communication between the EXU's to support the mapping of a wide range of algorithms and to ensure conflict-free data routing for efficient hardware utilization;
c) support for moderate (1–10) hardware multiplexing on the EXU's, for fast computation of tight inner loops;
d) support for low overhead branching between loop iterations;
e) wide instruction bandwidth;
f) wide I/O bandwidth (hundreds of megabytes/s).

The examples were drawn from real-time video, speech, and image processing applications. They included biquadratic filters (nonpipelined and pipelined [16], [21]), the RGB to YUV converter of [14], the 3 × 3 image convolver and nonlinear sorting filters from [18] and [19], a memory controller for video coding [20], a dynamic time warp speech processor [12], and the word

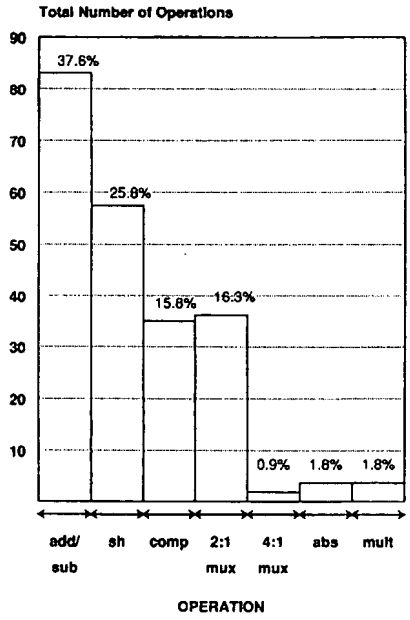


Fig. 1. Total number of operations versus operation type.

TABLE I
COMPUTATIONS AND I/O SUMMARY

	YUV Conv.	3 × 3 Conv.	Word Proc.	Grammar Proc.
MOPS	540	170	225/520	200
IO (MB/sec)	162	20	85/600	265

and grammar processing subsystems of a large vocabulary real-time speech recognition system [4], [22].

By careful analysis of these examples we were able to derive the essential features of our programmable architecture. For example, by profiling the total number of occurrences of various operations across the benchmark set, we were able to identify the critical operations that should be supported. The result is presented in Fig. 1. The percentage occurrence of all occurrences is also listed. Clearly, by adopting the ten-percent rule, architectural support for add/sub, shifts, comparisons, and 2-to-1 multiplexing is desirable.

Table I summarizes the computational and I/O requirements of a few examples from [4], [14], [18], [19], and [22]. From these numbers we can see that real-time DSP applications place a tremendous demand on both computation and bandwidth requirements.

B. Software-Configurable Hardware

Fig. 2 shows the relative flexibility and performance of implementation approaches available to the system designer. Software-based microprocessor and digital signal processor approaches are very flexible, but often do not achieve the required performance. ASIC approaches often have high nonrecurring engineering (NRE) costs, and can take considerable time (months) and effort to fabricate and test. Software-configurable hardware combines the flexibility of software approaches with the high performance of hardware approaches. We will now describe the PADDI software-configurable architecture created to fill this gap.

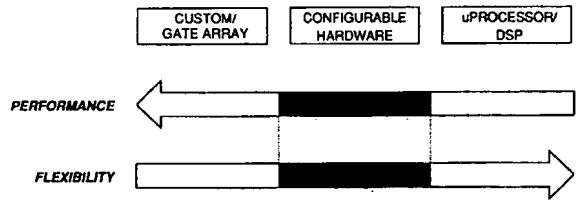


Fig. 2. Implementation alternatives.

II. PADDI Architecture and VLSI Implementation

A. Overview

The PADDI architecture provides a novel hardware platform for the rapid prototyping of algorithmic specific high-speed data paths. PADDI is software configurable, which allows algorithms to be hardwired into the architecture.

The basic architecture of our prototype chip is outlined in Fig. 3. It contains a cluster of eight EXU's, each with its own local controller. (The complete architecture contains four of these clusters on a single chip.) The EXU's are connected by a dynamically configurable, crossbar communication network. The architecture addresses the key requirements for rapid prototyping of dedicated high-speed data paths as follows. a) Each EXU contains dedicated hardware support for fast arithmetic. b) A crossbar switch that is under program control ensures conflict-free data routing within a cluster of EXU's. Global broadcasting from a single source is supported, and the dynamic nature of the interconnect ensures that multiple sources can be merged at a single destination. c) The combination of flexible local interconnect, distributed memory (in the form of register files), and local controllers supports direct mapping of flow graphs to EXU's, and the multiplexing of more than one operation onto a given EXU. d) By using multiple EXU's rather than superpipelining, a single or a few EXU's to achieve high computation rates, and by providing appropriate logic, low branch penalty for conditional branches and between loop iterations is achieved. e) High instruction bandwidth is ensured by assigning to each EXU its own dedicated controller. e) One hundred and twenty eight dedicated I/O pins allow EXU clusters to communicate with other clusters with high bandwidth (400 megabytes/s). The component parts of the architecture will be described in the following sections.

B. EXU Architecture

Fig. 4 shows the internal architecture of an EXU. Two 16-b-wide register files, each containing six registers, are used for the temporary buffering of data. The files are

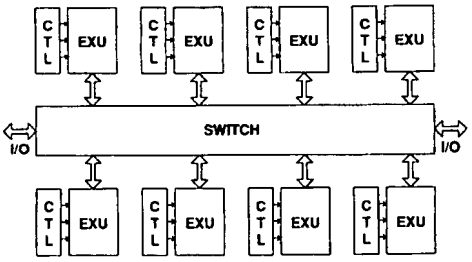


Fig. 3. Prototype architecture.

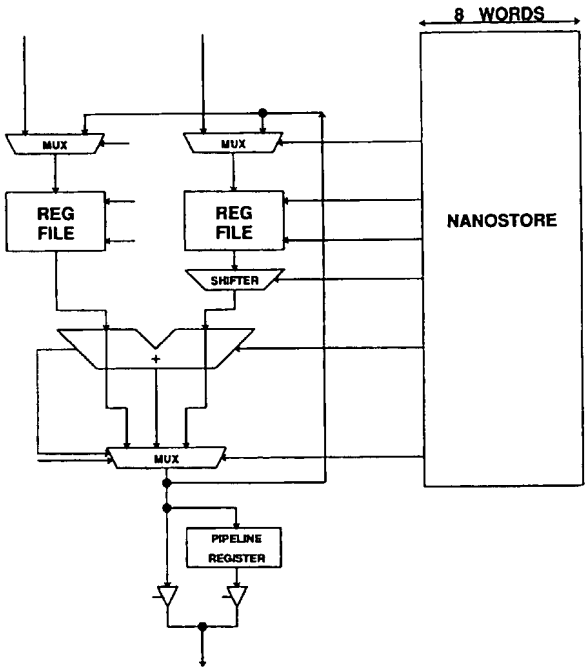


Fig. 4. EXU architecture.

dual-ported for simultaneous read and write operations. They can also be configured as delay lines to support pipelining and retiming of operations. In each register file, one of the six registers is configured as a scan register. It can be initialized to contain an arbitrary value (for the implementation of constant variables), read and written as a regular register, or used for scan testing. A fast carry-select adder and logarithmic shifter are used to implement the arithmetic functions, and hardwired logic provides single-cycle comparison, and maximum and minimum functions. Arithmetic can be performed in two's complement or unsigned format with automatic saturation. An optional pipeline register is available at the output of each EXU. By using the register, the user can increase the maximum sampling rate by overlapping EXU operations with data transmission over the network. This can be useful in applications where the additional latency has no negative effect. However, if the operation is in a feedback loop, the additional pipeline register would normally not be used. A status flag ($a \geq b$) is available to other EXU's and the external world to affect local and global program control flow, respectively. The EXU's normally provide 16-b accuracy, but two can be concatenated for increased 32-b accuracy. The register delay, type of arithmetic, output pipeline register, and EXU linking are controlled by mode bits set by the user.

C. Communication Network

To ensure flexible, conflict-free and high-bandwidth data routing, a crossbar network has been selected to interconnect the processors. This network routes both data as well as status flags. The data routing is under program control and can be changed in each program cycle; the routing of the status flags is static and set at compile time. Static flag routing was chosen as a reasonable compromise between flexibility and hardware efficiency.

The main challenge in the design of the crossbar network is to ensure a pitch matching between the crossbar switches and the EXU's. Therefore, a layered crossbar structure has been developed as shown in Fig. 5. A detail of the data-routing bit slice that connects EXU's A and E to each other, to other EXU's in the cluster, and the I/O buses is pictured. The layered switch implementation is organized as follows. The Type I switch connects the input of an EXU to either one of its neighbors (B, C, D for EXU A) or to the I/O buses or the other half of the cluster via a Type II switch. The squares and the circles represent inputs and outputs to the switches, respectively. A Type II switch is detailed. Data lines are run horizontally and control lines vertically. The major advantage of the proposed approach is that it allows all horizontal buses to fit within the pitch provided by the EXU's and hence save a substantial amount of area. Finally, in order to make the design even denser, the switches are implemented using NMOS pass transistors only. Weak PMOS feedback transistors restore the weak high level passed by the NMOS pass transistors and improve noise margin.

The problems of sizing the feedback transistors for both Type I and II switches were coupled to avoid using extra decoupling buffers. Fig. 6 shows a circuit diagram of both switches. Fig. 7 shows SPICE switching waveforms of nodes A, B, and C of Fig. 6 with W/L of the feedback PMOS transistors as a parameter. In the case where W/L is zero, no feedback transistor is present, and node B is never pulled to rail. In the case where W/L is 8/2, node C is never pulled to ground because the inverter driving node C is not strong enough to counter the feedback transistor of the Type I switch. A reasonable compromise is to choose $W/L = 3/3$.

D. Control

Each EXU requires a 53-b horizontal control word, and so the overall instruction bandwidth for all eight EXU's is quite high. In order to simultaneously achieve both a high instruction and data bandwidth, the control strategy shown in Fig. 8 was used. At run time, an external sequencer broadcasts a 3-b global instruction to each EXU,

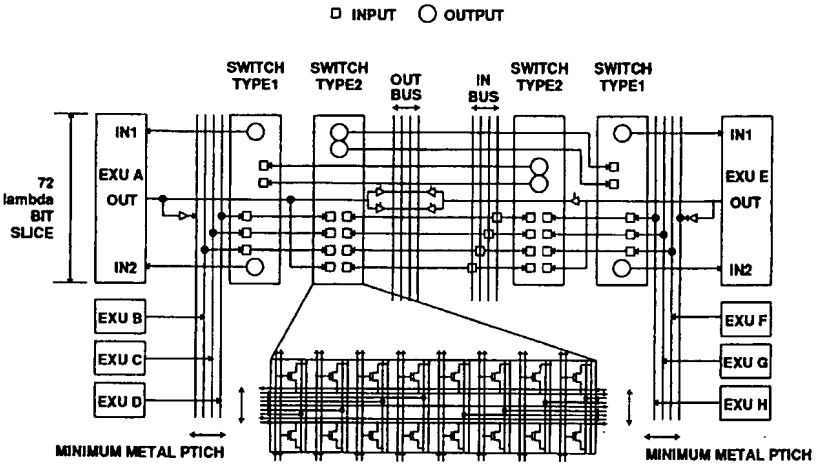


Fig. 5. Simplified schematic of crossbar switch.

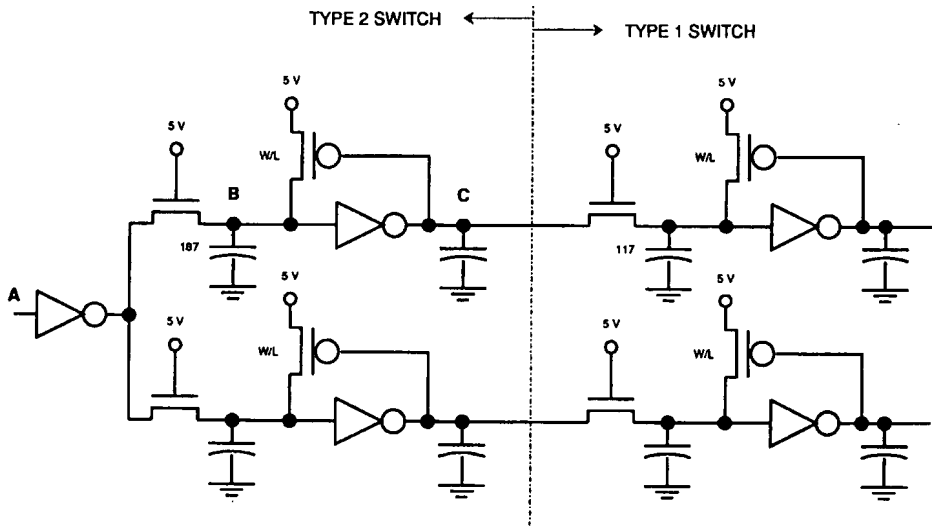


Fig. 6. Type I and Type II switches with weak PMOS pull-ups.

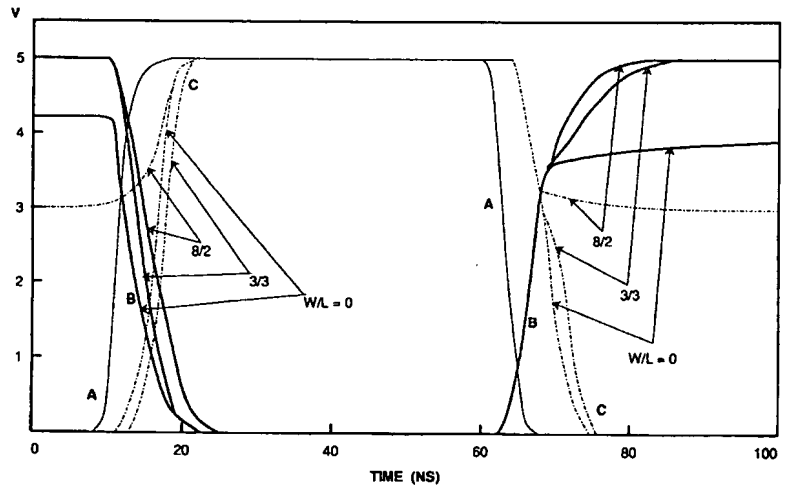


Fig. 7. Type I and II switching waveforms.

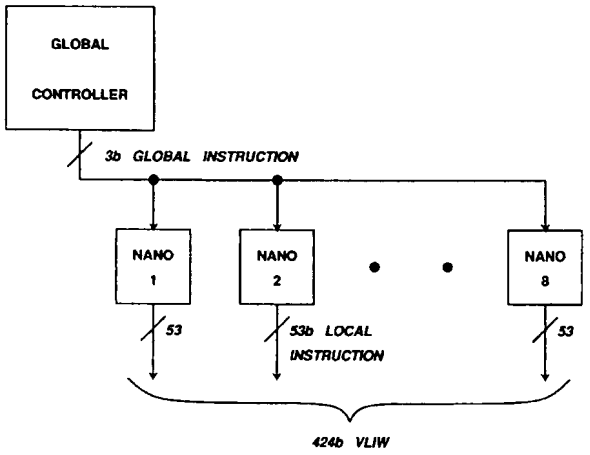


Fig. 8. Basic control strategy.

which is locally decoded into a 53-b instruction word. In this fashion, a 3-b word is used to specify a 424-b very long instruction word (VLIW). The architecture is SIMD in that each EXU receives the same global instruction, but MIMD in that each decoded instruction is unique to the associated EXU.

Status information can be communicated between the EXU's and the external controller to affect both the local and global control flows. Each EXU instruction contains two interrupt state fields. By setting these, the EXU can accept interrupts from other EXU's or the external world and then vector to the apropos address contained in its precompiled interrupt vectors.

The decoders of each EXU are SRAM-based nanostores that are configured at setup time. Fig. 9 shows a section of the SRAM. Each SRAM contains eight words, which allows eight different operations to be multiplexed on its associated EXU. Master–slave scan latches at its I/O are connected to a global serial shift register (scan chain) to allow serial configuration of the SRAM. The SRAM is implemented using a conventional six-transistor cell and operates with two-phase nonoverlapping clocks.

E. Configuration

All chip configuration registers (e.g., constants, mode bits, interrupt vectors) and the SRAM scan latches are connected as a serial shift register. Only a few pins are needed to configure the chip using this scheme. Eight scans, one for each word, are required to completely load the nanostores. Two on-board FSM's generate the necessary clock and interface and internal control signals that allows the chip to boot directly from standard EPROM's without the need for additional glue logic. One FSM generates divided down scan clocks which allows the use of standard (slow) EPROM's for booting. The other generates control signals for scanning and writing each individual line, and for verifying the contents of the nanostores. On-board counters indicate when a scan is completed and keep track of the current nanostore word being written.

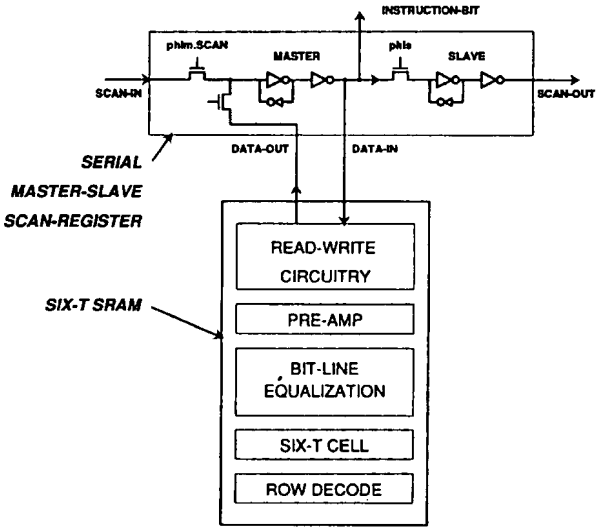


Fig. 9. SRAM schematic.

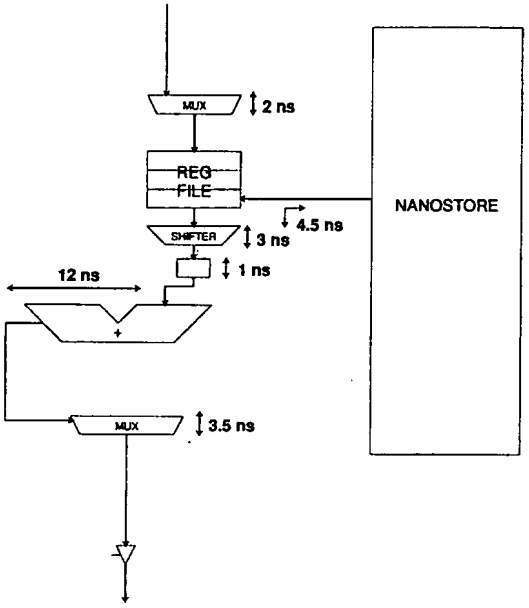


Fig. 10. EXU critical path.

III. Simulation and Testing

A. Simulation

SPICE simulations were performed to simulate the critical path of the chip. The respective load capacitances were estimated from the worst-case IC process parameters and incorporated into the SPICE decks. The critical path simulation results for an EXU and a four-quadrant chip are shown in Figs. 10 and 11 respectively. The units shown are in nanoseconds. The critical path begins from the issue of a read address to register file *B*. A delay of 24 ns is incurred during EXU transit from the register file, through the shifter and inversion logic, through the carry path of the carry select adder, and through the saturation

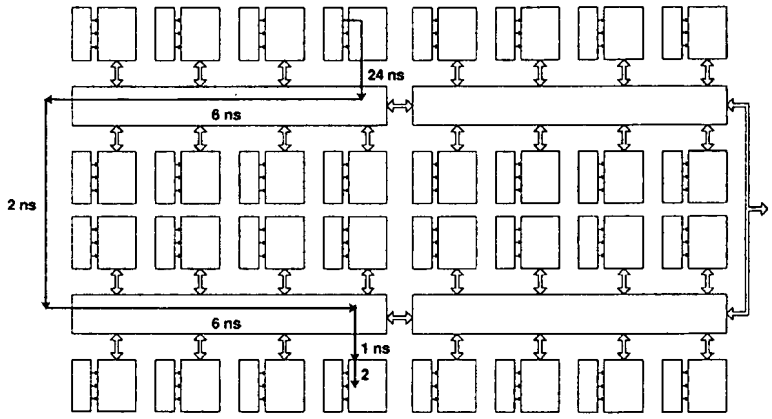


Fig. 11. Four-quadrant critical path.

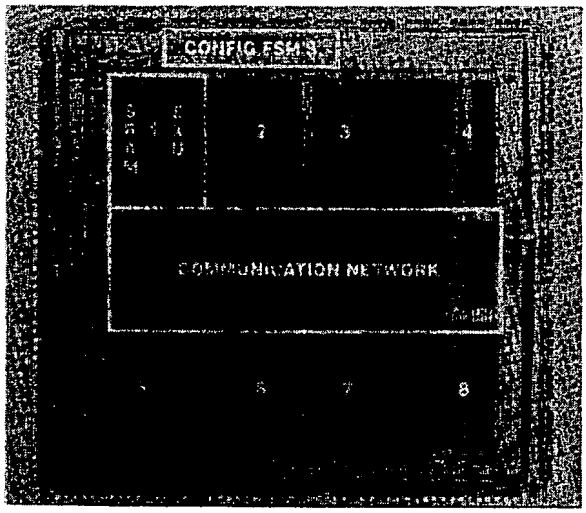


Fig. 12. Chip photograph.

TABLE II
CHIP CHARACTERISTICS

EXU's	8 units 16-32 b
Register Files	2 files, Six 16-b registers
Nanostores	53 b 8 words
I/O Ports	128
Clock Frequency	25 MHz
Compute Power	200 MIPS
I/O Bandwidth	400 megabytes/s
No. of Transistors	140 106
Die Size	8.8 × 9.5 mm²

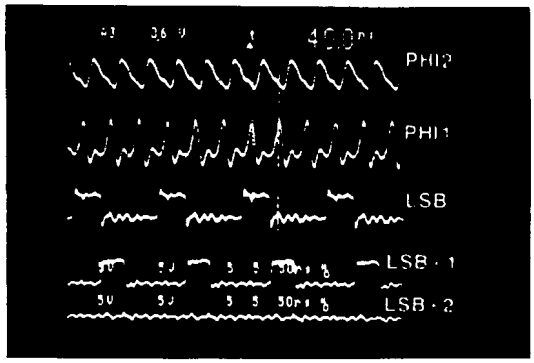


Fig. 13. 25-MHz mod 3 counter.

logic to the output of the EXU. An additional 15 ns is lost during transit through the crossbar networks, after which 2 ns are required to latch the data into the input of the target EXU. The total simulated critical delay is 41 ns.

The chip was implemented in 1.2-μm CMOS technology and tested and was fully functional on first silicon. Fig. 12 shows a photograph of the chip and Table II summarizes the chip characteristics.

B. Testing

Fig. 13 shows the assembly code and scope trace for a mod 3 counter that cycles between 0, 1, and 2 at 25 MHz.

We note that the prototype chip, which contains a single quadrant, runs at a maximum clock frequency of 25 MHz with a critical path delay of 40 ns. This indicates that there is excellent agreement with the SPICE simulations. The only major difference between a single quadrant and one with four quadrants is the additional inter-quadrant transit time which, with proper buffering, can be limited to 1 or 2 ns.

The SFG of a low-pass biquadratic filter is shown in Fig. 14. The multiplying coefficients were converted to a canonic signed digit format to minimize the number of nonzero bits and transformed into shifts and adds (Fig. 15). A processor schedule using three EXU's and three instructions is shown in Fig. 16. Fig. 17 shows the assembly code (mapped to three units and three instructions). Fig. 18(a) and (b) shows plots of the acquired impulse response and the corresponding frequency response, respectively. The arithmetic mode is 16-b two's complement and the impulse is input at bit 13. The measured results agree well with simulations. Due to limitations of

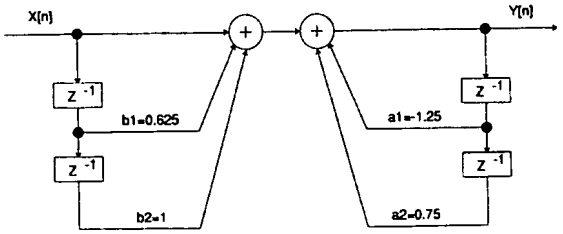


Fig. 14. Simple low-pass biquadratic filter.

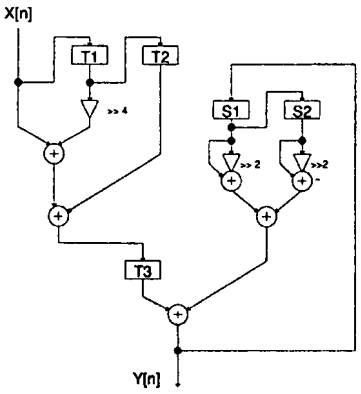


Fig. 15. Transformed biquad.

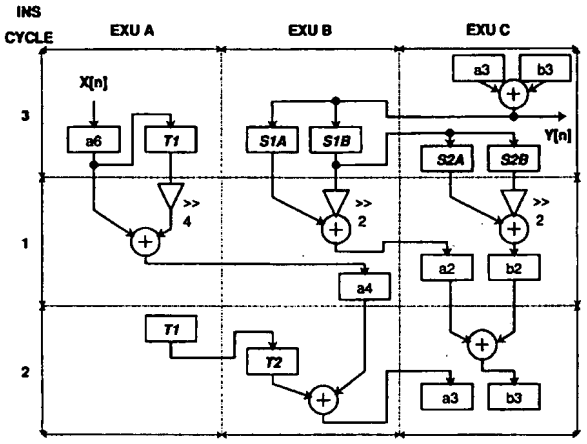


Fig. 16. Biquad processor schedule.

the signal analyzer in acquiring data, the maximum clock rate of this biquad was constrained to 10 MHz.

Table III shows several other benchmarks which can be implemented on this architecture using several PADDI chips.

IV. Programming Tools for PADDI

The low-level programming tools, the `pas` assembler and `psim` simulator, provide the foundation for the higher-level synthesis tools.

```
DEFAULTS {
   A6 = 0, B6 = 0, SIGNED, UNLINK
   NORMAL_A, NORMAL_B,
   BFSW=11111111b,
   IEN1 = 0, IEN2 = 0}

EXU_A{
   1: (A6 + (B6 >> 4));
   2: (B6);
   3: A6 = I2L, B6 = EXU_A, (A6);}

EXU_B{
   1: A4 = EXU_A, (A6 + B6 >> 2);
   2: B6 = EXU_A, (A4 + B6);
   3: A6 = EXU_C, B6 = EXU_C, (B6);}

EXU_C{
   1: A2 = EXU_B, B2 = EXU_C, (A6-(B6>>2));
   2: A3 = EXU_B, B3 = EXU_C, (A2 - B2);
   3: A6 = EXU_B, B6 = EXU_B, (A3 + B3), O1;}
```

Fig. 17. Biquad assembly code.

A. The Pas Assembler

`Pas` represents the lowest software-level interface between the programmer and the PADDI architecture, providing a method for describing algorithms. The `pas` assembly language was designed and implemented with the interconnection network of the PADDI architecture in mind: programs written in it can easily exploit intercommunication between execution units. The intercommunication follows a "receiver-controlled" model in which the receiving unit controls the routing of the actual communication while the broadcasting unit only concerns itself with the data or flag to be communicated (except when broadcasting to the external world; in this case the broadcaster must specify which output bus to employ). In addition to being able to express all available PADDI operations in a convenient C-like syntax, the assembler also allows for the explicit specification of instructions within the nanostores at the individual bit level.

B. The Psim Simulator

`Psim` serves as a tool for simulating and debugging multiple-chip PADDI algorithms in software. It consists of a simulation engine coupled with an X-based graphical user interface (GUI). The simulation engine can operate both as a "black box," allowing it to interface with external software tools, or as a stand-alone simulation environment when coupled with the X-based GUI. The stand-alone simulation environment supports many of the common debugging features, including single-stepped execution and the ability to modify registers and instructions "on the fly."

C. Software Compilation

An automated compilation path (Fig. 19) from a high-level data flow language Silage [15] to the PADDI chip, which includes partitioning, scheduling, and code gener-

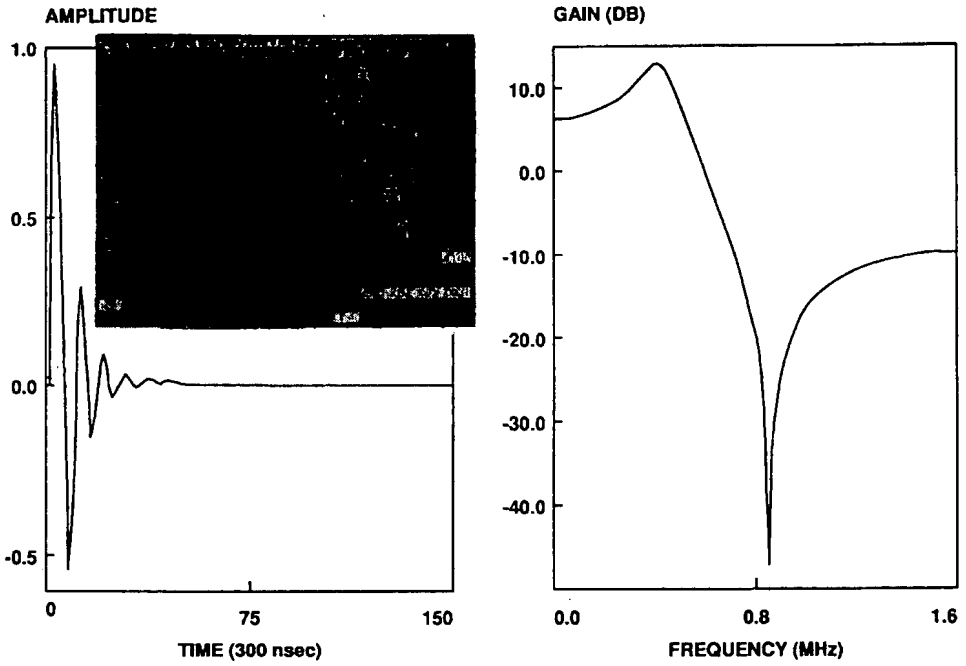


Fig. 18. Biquad impulse response.

TABLE III
BENCHMARKS

BENCHMARK	Possible Sampling Rate	EXU's Required
3 × 3 Linear Convolver (Image processing)	25 MHz	11
3 × 3 Nonlinear Sorting Filter (Image processing)	25 mHz	16
RGB Video Matrix Converter	25 MHz	31
Flexible Memory Control Chip for Video Coding	25 MHz	28
Biquad Direct Form II (time-multiplexed)	5 MHz	9
Biquad Direct Form II (pipelined)	25 MHz	55

ation, is currently under development. A paper detailing the compiler design has been submitted to [6].

V. Limitations of Other Approaches

The number of choices for FPGA's is numerous with offerings from XILINX [11], Actel [1], Plus Logic [17], Plessey [9], Algotronix [8], ATT (10), and others. Due to their bit-level granularity, these FPGA's will not support as flexible routing of wide data buses and will not have as fast adders (for the same technology) as a word-level granular architecture with flexible bus interconnections and adders optimized for speed. FPGA's also do not typically support hardware multiplexing of their CLB's. In software-configurable FPGA's, the functions of the CLB's can be redefined, but this is not typically done in high-speed applications since reconfiguration time is of the order of milliseconds. In order to illustrate these points, we have mapped several benchmarks to the popular XILINX XC3090 family.

The results are shown in Table IV, case A. The FPGA speed numbers are optimistic because no account is taken for routing delays. Because of the ability to have faster arithmetic for the same technology, more flexible interconnectivity of data buses, support for hardware multiplexing, and more efficient implementation of register files, PADDI is better suited for data-path-intensive applications.

When we started our investigations only the XC3000 family was available. The recently introduced XILINX XC4000 series, which uses 0.8- and 0.5-μm CMOS technology and which has hooks for faster adders and a different interconnect architecture, will affect the above comparison.

As a further comparison, a Motorola DSP56000 can operate at 10.25 MIPs and has a data I/O bandwidth of 60 megabytes/s. For video sampling rates, this translates to roughly two available instructions per sample. This type of limited performance and I/O capability clearly limits the applicability of general-purpose DSP's to real time DSP applications.

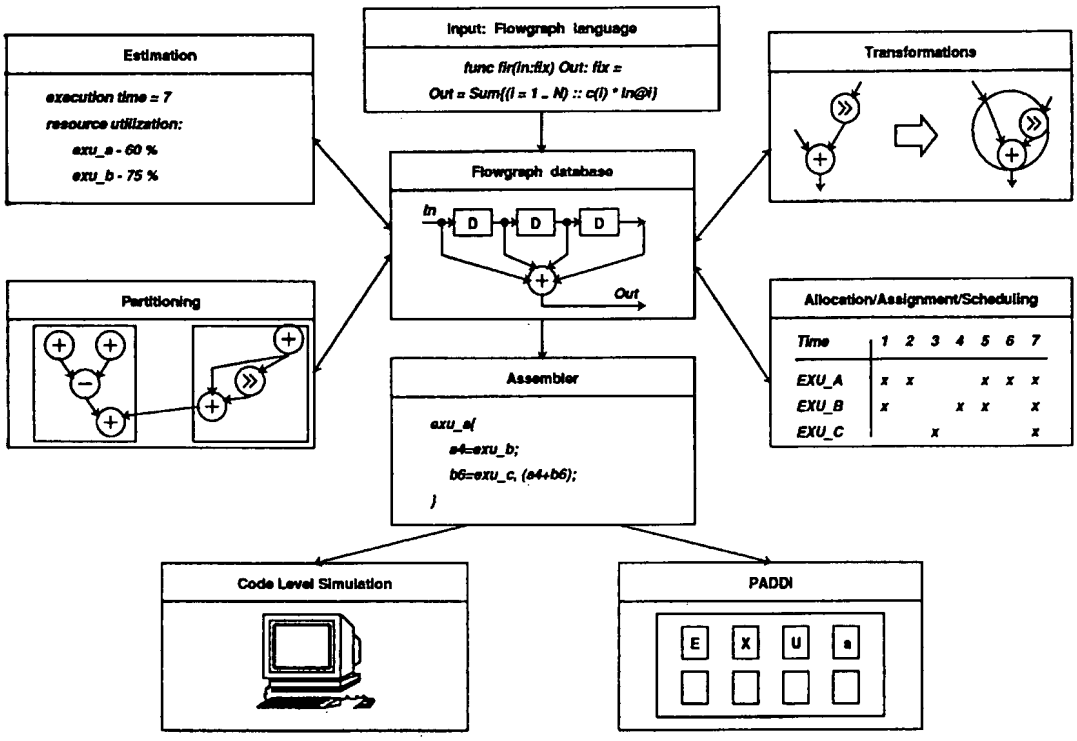


Fig. 19. Software environment.

TABLE IV
COMPARISON OF XILINX AND PADDI

Case	XILINX Min Sampl Int (ns)	XILINX No. of CLB's	PADDI Min Sampl Int (ns)	PADDI No. of EXU's
16-b biquad (case A)	153	176	75	4
16-b biquad (case B)	190	400	200	9
16-b biquad (case B, pipelined)	144	1504	40	55
3 × 3 Linear Convolver	144	— (3 chips)	40	11

Video signal processors (VSP's) [2], [3], [7], [13], [23] usually have a few complex and highly pipelined arithmetic logic units with on-board support for memory, and are not designed to support conditional branching efficiently. The example of [23] can operate three 12-b execution units at 27 MHz (81 MIPs) with a data I/O bandwidth of 405 megabytes/s, but typically operates at 13.5 MHz (40 MIPs) due to the latency of the long pipelines. The chip presented here can operate eight 16-b execution units at 25 MHz (200 MIPs) with a data I/O bandwidth of 400 megabytes/s. Because of the larger degree of concurrency due to the smaller level of granularity of the EXU's, and smaller branch penalty, PADDI is better suited for data-path prototyping.

VI. Conclusion

The architecture and implementation of a reconfigurable multiprocessor IC for rapid prototyping of real-time data paths has been described. The chip targets high-performance digital signal processing applications. A 16 EXU (400 MIPs) processor is currently under design, together with a multichip module approach, which could support up to 32 EXU's (800 MIPs) in a single package.

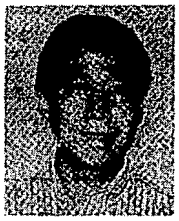
Acknowledgment

The authors thank Prof. R. W. Brodersen, A. Lee, S. Li, E. Ng, D. Schultz, C. Yu, and the members of the BJ-Group for their support.

References

[1] M. Ahrens *et al.*, "An FPGA family optimized for high densities and reduced routing delay," in *Proc. Custom Integrated Circuits Conf.* May 1990, pp. 31.5.1–4.
[2] K. Aono, M. Toyokura, and T. Araki, "A 30NS (600 MOPS) image processor with a reconfigurable pipeline," in *Proc. Custom Integrated Circuits Conf.*, May 1989.
[3] T. Baji *et al.*, "A 20-ns CMOS DSP core for video-signal processing," *IEEE J. Solid-State Circuits*, vol. 23, pp. 156–157, Feb. 1988.
[4] D. C. Chen, R. Yu, R. W. Brodersen, and J. Rabaey, "A VLSI grammar processing subsystem for a real time large vocabulary continuous-speech recognition system," in *Proc. Custom Integrated Circuits Conf.*, May 1990, pp. 13.3.1–5.
[5] D. C. Chen and J. M. Rabaey, "PADDI: Programmable arithmetic devices for digital signal processing," in *VLSI Signal Processing IV*. New York: IEEE Press, Nov. 1990, pp. 240–249.
[6] D. C. Chen *et al.*, "An integrated system for rapid prototyping of

high performance algorithm specific data paths," submitted to the ASAP 92 Int. Conf. Application-Specific Array Processors.
[7] T. Fukushima, "A survey of image processing LSIs in Japan," in *IEEE Int. Proc. Conf. Pattern Recognition*, vol. 2, 1990, pp. 394-401.
[8] J. P. Gray and T. A. Kean, "Configurable hardware: A new paradigm for computation," in *Advanced Research in VLSI, Proc. Decennial Caltech Conf. VLSI*, Mar. 1989, pp. 279-295.
[9] N. Hastie and R. Cliff, "The implementation of hardware subroutines on field programmable gate arrays," in *Proc. Custom Integrated Circuits Conf.*, May 1990, pp. 31.4.1-4.
[10] D. Hill and D. Cassiday, "Preliminary description of Tabula Rosa: An electrically configurable hardware design," in *Proc. ICCD*, Sept. 1990, pp. 391-395.
[11] H. C. Hsieh *et al.*, "A second generation user-programmable gate array," in *Proc. Custom Integrated Circuits Conf.*, May 1989.
[12] R. Kavaler, "The design and evaluation of a speech workstation," Univ. of California, Berkeley, Tech. Rep. Memo. UCB/ERL M86/39, 1986.
[13] T. Minami *et al.*, "A 300 MOPS video signal processor with a parallel architecture," in *ISSCC Dig. Tech. Papers*, Feb. 1991, pp. 252-253.
[14] S. Note, J. Van Meerbergen, F. Catthor, and H. de Man, "Hardwired data path synthesis for high speed DSP systems with the Cathedral III compilation environment," in *Logic and Architecture Synthesis for Silicon Compilers*. Amsterdam: Elsevier Science (North-Holland), Feb. 1989, pp. 243-254.
[15] P. Hilfinger, "A high level language and silicon compiler for digital signal processing," in *Proc. IEEE Custom Integrated Circuits Conf.*, May 1985, pp. 240-243.
[16] K. K. Parhi and D. G. Messerschmitt, "Pipeline interleaving and parallelism in recursive digital filters, I and II," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 37, pp. 1099-1134, July 1989.
[17] *FPGA2040*, Plus Logic, San Jose, CA, 1989.
[18] P. Ruetz and R. Brodersen, "A realtime image processing chip set," in *ISSCC Dig. Tech. Papers, Conf.*, Feb. 1986, pp. 148-149.
[19] P. A. Ruetz, "Architectures and design techniques for real-time image processing ICs," Univ. of California, Berkeley, Tech. Rep. Memo. UCB/ERL M86/37, 1986.
[20] R. Schmidt, "A memory control chip for formatting data into blocks suitable for video coding applications," *IEEE Trans. Circuits Syst.*, vol. 36, pp. 249-258, Oct. 1989.
[21] M. A. Soderstrand and B. Sinha, "Comparison of three new techniques for pipelining IIR digital filters," in *Proc. Asilomar Conf. Circuits Syst.*, 1985, pp. 439-443.
[22] A. Stölzle, "A real time large vocabulary speech recognition system," Ph.D. dissertation, Univ. of California, Berkeley, May 1992.
[23] A. H. van Roermund *et al.*, "A general purpose programmable video signal processor," *IEEE Trans. Consumer Electron.*, vol. 35, pp. 249-258, Aug. 1989.



Dev C. Chen (S'80-M'83-S'85-M'91) was born in Kingston, Jamaica, on December 16, 1957. He received the B.S. degree in physics and electronic engineering from the University of Manchester, U.K., and the M.S. degree from Yale University, New Haven, CT, in 1979 and 1981, respectively. In 1992 he received the Ph.D. degree in electrical engineering and computer sciences from the University of California at Berkeley.

From 1981 to 1985 he was with the Hewlett Packard Laboratories, Palo Alto, CA, where he engaged in the research and development of submicrometer MOS devices, and was the co-author of several patents. In 1986 he returned to graduate school where he designed a grammar processor for a real-time large vocabulary speech recognition system. His thesis work concentrated on the design of architectures for the rapid prototyping of high-speed digital signal processing systems. In 1992 he joined SUN Microsystems Inc., Mountain View, CA, where he is involved with the design of an advanced high-performance microprocessor. His current interests are in microprocessor design, caches for multiprocessors, and computer systems architecture.

Jan M. Rabaey (S'80-M'83) was born in Veurne, Belgium, on August 15, 1955. He received the E.E. and Ph.D. degrees in applied sciences in 1978 and 1983, respectively, from the Katholieke Universiteit, Leuven, Belgium, where he worked on computer-aided design tools for switched capacitor circuits.

From 1983 to 1985 he was associated with the University of California, Berkeley, as a Visiting Research Engineer, where he developed an automated synthesis system for multiprocessor DSP architectures. From 1985 to 1987 he directed the Architectural and Algorithmic Strategies group of the Design Methodologies for VLSI System Division at IMEC, Belgium. In 1987 he joined the faculty of the University of California, Berkeley, as an Assistant Professor. His current interests are in computer-aided analysis and automated design of digital signal processing circuits, and architectural synthesis.